Software Testing and Verification

Problem Set 5: Axiomatic Verification – Solution Notes

1. a. would not: pre-condition not satisfied
   b. would not: cannot determine if Q holds in this case or not since FINAL value of z is not given
   c. would not: program does not terminate
   d. would: y<z => Q is false
   e. would not: the FINAL value of z may be such that Q holds
   f. would not: Q will hold in this case regardless of the initial value of z
   g. would not: Q will hold in this case when P holds initially, or when z=0 initially
   h. would not: Q will hold in this case regardless of the initial value of z

2.

```
            {x>y}
              temp := x
            {temp=x ∧ x>y}
              x := y
            {x=y ∧ temp>y ∧ temp>x}
              y := temp
            {y=temp ∧ x=y' ∧ temp=x' ∧ x'>y'} => {y=temp ∧ temp>x}
              if temp>z then
                y := z
                z := temp
                  if x>y then
S1                  temp := x
                    x := y          S2
                    y := temp
                  end_if
              end_if
            {x≤y≤z}
```

$\{y=temp \wedge temp>x\}$ if temp>z then S1 $\{x \le y \le z\}$

Using the if-then ROI, we need to show:

(1) $\{y=temp \wedge temp>x \wedge temp>z\}$ S1 $\{x\le y\le z\}$?
(2) $(y=temp \wedge temp>x \wedge temp\le z) \Rightarrow x<y\le z \Rightarrow Q$ √

For (1) above we have: $\{y=temp \wedge temp>x \wedge temp>z\}$

y := z

$\{y=z \wedge y'=temp \wedge temp>x \wedge temp>z\}$

z := temp

$\{z=temp \wedge y=z' \wedge y'=temp \wedge temp>x \wedge temp>z'\} \Rightarrow \{z=temp \wedge temp>x \wedge temp>y\}$

if x>y then S2

$\{x\le y\le z\}$?

Using the if-then ROI a second time, we need to show:

(3) {z=temp ∧ temp>x ∧ temp>y ∧ x>y} S2 {x≤y≤z} ?
(4) (z=temp ∧ temp>x ∧ temp>y ∧ x≤y) => x≤y<z => Q √

For (3) above we have:

{z=temp ∧ temp>x ∧ temp>y ∧ x>y}
temp := x
{temp=x ∧ z=temp' ∧ temp'>x ∧ temp'>y ∧ x>y} => {temp=x ∧ z>x ∧ z>y ∧ x>y}
x := y
{x=y ∧ temp=x' ∧ z>x' ∧ z>y ∧ x'>y} => {x=y ∧ z>temp ∧ z>y ∧ temp>y}
y := temp
{y=temp ∧ x=y' ∧ z>temp ∧ z>y' ∧ temp>y'} => {y=temp ∧ z>temp ∧ z>x ∧ temp>x}
=> {x<y<z} => Q √

3. Let I = (Found ∧ Key=List[Index]) V (~Found ∧ ∀ Index < i ≤ N, key<>List[i])

INITIALIZATION: Does P => I?

P = N≥1 ∧ ~Found ∧ Index=N
=> (~Found ∧ ∀ Index < i ≤ N, key<>List[i])
So P => I.

PRESERVATION: Does {I ∧ b} s {I}?

I ∧ b = {[(Found ∧ Key=List[Index]) V (~Found ∧ ∀ Index < i ≤ N, Key<>List[i])] ∧ Index>0 ∧ ~Found)]}

= {(~Found ∧ ∀ Index < i ≤ N, Key<>List[i]) ∧ Index>0)}

To show: {I ∧ b}

```
if Key=List[Index] then
   Found := true
else
   Index := Index-1
End_if_else

   {I}
```

we must, by the if-then-else Rule of Inference, show:

(1): {I ∧ b ∧ Key=List[Index]} Found := true {I}, and
(2): {I ∧ b ∧ Key<>List[Index]} Index := Index-1 {I}

For (1) we have:

$\{\sim\text{Found} \wedge (\forall\ \text{Index} < i \leq N, \text{Key}<>\text{List}[i]) \wedge \text{Index}>0 \wedge$
$\text{Key}=\text{List}[\text{Index}]\}$

Found := true

$\{\text{Found} \wedge (\forall\ \text{Index} < i \leq N, \text{Key}<>\text{List}[i]) \wedge$
$\text{Key}=\text{List}[\text{Index}] \wedge \text{Index}>0\} \Rightarrow I$

For (2) we have:

$\{\sim\text{Found} \wedge (\forall\ \text{Index} < i \leq N, \text{Key}<>\text{List}[i]) \wedge \text{Index}>0 \wedge$
$\text{Key}<>\text{List}[\text{Index}]\}$

Index := Index-1

$\{\sim\text{Found} \wedge (\forall\ \text{Index} < i \leq N, \text{Key}<>\text{List}[i]) \wedge \text{Index}\geq 0\} \Rightarrow I$

So $\{I \wedge b\}\ s\ \{I\}$.

FINALIZATION: Does $(I \wedge \sim b) \Rightarrow Q$?

$I \wedge \sim b = \{[(\text{Found} \wedge \text{Key}=\text{List}[\text{Index}]) \vee (\sim\text{Found} \wedge \forall\ \text{Index} < i \leq N,$
$\text{Key}<>\text{List}[i]) \wedge (\text{Index}\leq 0 \vee \text{Found})]\}$

$\Rightarrow \{[(\text{Found} \wedge \text{Key}=\text{List}[\text{Index}]) \vee (\sim\text{Found} \wedge \forall\ 1 \leq i \leq N,$
$\text{Key}<>\text{List}[i]) = Q$

4.

ROI: $$\frac{\{P\}\ s\ \{I\},\ \{I \wedge \sim b\}\ s\ \{I\},\ (I \wedge b) \Rightarrow Q}{\{P\}\ \text{repeat}\ s\ \text{until}\ b\ \{Q\}}$$

Let I =

$[(\text{Found} \wedge \text{Key}=\text{List}[\text{Index}]) \vee (\sim\text{Found} \wedge \forall\ i \in [1,\text{First}) \cup (\text{Last},N], \text{key}<>\text{List}[i])]$
$\wedge$ iorder

INITIALIZATION: Does $\{P\}\ s\ \{I\}$?

$\{N\geq 1 \wedge \text{iorder} \wedge \text{First}=1 \wedge \text{Last}=N \wedge \sim\text{Found}\}$

Index := (First + Last) div 2

$\{N\geq 1 \wedge \text{iorder} \wedge \text{First}=1 \wedge \text{Last}=N \wedge \sim\text{Found} \wedge \text{Index}=(1+N)\text{div}2\} = P1$

To show: {P1}

```
if Key=List[Index] then
   Found := true
else
   if Key<List[Index] then
      First := Index+1
   else
      Last := Index-1
   end-if-else
end-if-else
```

{I}

we must show:

(1): {P1 ∧ Key=List[Index]} Found := true {I}, and
(2): {P1 ∧ Key<>List[Index]} if...then...else... {I}

For (1) we have:

{N≥1 ∧ iorder ∧ First=1 ∧ Last=N ∧ ~Found ∧ Index=(1+N)div2 ∧ Key=List[Index]}

Found := true

{N≥1 ∧ iorder ∧ First=1 ∧ Last=N ∧ Found ∧ Index=(1+N)div2 ∧ Key=List[Index]} => Found ∧ Key=List[Index] ∧ Iorder => I

For (2) we must show:

(a) {P1 ∧ Key<List[Index]} First := Index+1 {I}, and
(b) {P1 ∧ Key>List[Index]} Last := Index-1 {I}

For (a) we have:

{N≥1 ∧ iorder ∧ First=1 ∧ Last=N ∧ ~Found ∧ Index=(1+N)div2 ∧ Key<List[Index]}

First := Index+1

{N≥1 ∧ iorder ∧ First=(1+N)div2+1 ∧ Last=N ∧ ~Found ∧ Index=(1+N)div2 ∧ Key<List[Index]} =>

{~Found ∧ ∀ i ∈ [1, (1+N)div2+1) U (Last,N], key<>List[i]) ∧ iorder} =

{~Found ∧ ∀ i ∈ [1,First) U (Last,N], key<>List[i]) ∧ iorder} => I

For (b) we have:

{N≥1 ∧ iorder ∧ First=1 ∧ Last=N ∧ ~Found ∧ Index=(1+N)div2 ∧ Key>List[Index]}

Last := Index-1

{N≥1 ∧ iorder ∧ First=1 ∧ Last=(1+N)div2-1 ∧ ~Found ∧ Index=(1+N)div2 ∧ Key>List[Index]} =>

{~Found ∧ ∀ i ∈ [1,First U ((1+N)div2-1,N], key<>List[i]) ∧ iorder} =

{~Found ∧ ∀ i ∈ [1,First) U (Last,N], key<>List[i]) ∧ iorder} => I

PRESERVATION: Does {I ∧ ~b}s{I}?

{[(Found ∧ Key=List[Index]) V (~Found ∧ ∀ i ∈ [1,First) U (Last,N], key<>List[i])] ∧ iorder ∧ ~(Found or First>Last)}

Index := (First + Last) div 2

{(~Found ∧ ∀ i ∈ [1,First) U (Last,N], key<>List[i]) ∧ iorder ∧ First<=Last ∧ Index=(First+Last)div2} = P2

To show: {P2}

```
if Key=List[Index] then
   Found := true
else
   if Key<List[Index] then
      First := Index+1
   else
      Last := Index-1
   end-if-else
end-if-else
```

{I}

we must show:

(1): {P2 ∧ Key=List[Index]} Found := true {I}, and
(2): {P2 ∧ Key<>List[Index]} if...then...else... {I}

For (1) we have:

$\{(\sim Found \wedge \forall\, i \in [1,First)\ U\ (Last,N],\ key<>List[i]) \wedge iorder \wedge$
$First<=Last \wedge Index=(First+Last)div2 \wedge Key=List[Index]\}$

Found := true

$\{(\sim Found \wedge \forall\, i \in [1,First)\ U\ (Last,N],\ key<>List[i]) \wedge iorder \wedge$
$First<=Last \wedge Index=(First+Last)div2 \wedge Key=List[Index] \wedge Found\} =>$

$\{Found \wedge Key=List[Index] \wedge iorder\} => I$

For (2) we must show:

(a) $\{P2 \wedge Key<List[Index]\}$ First := Index+1 $\{I\}$, and
(b) $\{P2 \wedge Key>List[Index]\}$ Last := Index-1 $\{I\}$

For (a) we have:

$\{(\sim Found \wedge \forall\, i \in [1,First)\ U\ (Last,N],\ key<>List[i]) \wedge iorder \wedge$
$First<=Last \wedge Index=(First+Last)div2 \wedge Key<List[Index]\}$

First := Index+1

$\{(\sim Found \wedge \forall\, i \in [1,First)\ U\ (Last,N],\ key<>List[i]) \wedge iorder \wedge$
$Key<List[Index] => I$

For (b) we have:

$\{(\sim Found \wedge \forall\, i \in [1,First)\ U\ (Last,N],\ key<>List[i]) \wedge iorder \wedge$
$First<=Last \wedge Index=(First+Last)div2 \wedge Key>List[Index]\}$

Last := Index-1

$\{(\sim Found \wedge \forall\, i \in [1,First)\ U\ (Last,N],\ key<>List[i]) \wedge iorder \wedge$
$Key>List[Index]\} => I$

FINALIZATION: Does $(I \wedge b) => Q$?

$I \wedge b = [(Found \wedge Key=List[Index])\ OR\ (\sim Found \wedge \forall\, i \in [1,First)\ U$
$(Last,N],\ key<>List[i])] \wedge iorder \wedge (Found\ or\ First>Last)$

$= [(Found \wedge Key=List[Index])\ OR\ (\sim Found \wedge \forall\, i \in [1,First)\ U$
$(Last,N],\ key<>List[i])] \wedge iorder \wedge Found \wedge First<=Last$

= (Found ∧ Key=List[Index] ∧ iorder ∧ First<=Last) => Q

OR

[(Found ∧ Key=List[Index]) OR (~Found ∧ ∀ i ∈ [1,First) U (Last,N], key<>List[i])] ∧ iorder ∧ ~Found ∧ First>Last

= (~Found ∧ V 1<=i<=N, key<>List[i] ∧ iorder ∧ First>Last) => Q

OR

[(Found ∧ Key=List[Index]) V (~Found ∧ ∀ i ∈ [1,First) U (Last,N], key<>List[i])] ∧ iorder ∧ Found ∧ First>Last

= (Found ∧ Key=List[Index] ∧ iorder ∧ First>Last)

=> (Found ∧ Key=List[Index]) V (~Found ∧ ∀ 1<=i<=N, key<>List[i])) = Q

5. a. The program will terminate for *ANY* initial values of integer variables x and y.

b. No. If $y_0 \geq 0$, the predicate "y<0" evaluates to false and the program terminates immediately. However, if $y_0<0$, y does not satisfy the "monotonically increasing" condition for a measure bounded from above (by the predicate "y<0") since x_0 may not be positive. A simple generalization of the Method stated in class that would allow its use in this case would be:

...identify a *measure* based on one or more program variables that satisfies the following properties:

1. monotonically decreases (or increases) with each iteration **after a finite number of initial iterations,**
2. is bounded from below (or above), and
3. can assume only a finite number of values before reaching the bound.

6.

a.
$$\frac{P \Rightarrow (\sim b \wedge Q)}{\{P\}\ \text{while } b \text{ do } s\ \{Q\}}\ ?$$

The rule is valid, since the antecedent implies that whenever the pre-condition, P, holds, the false branch will be executed and Q holds. The rule could be employed, for example, to prove:

{x=17} while x<0 do x := 0 {x>0}

b.

$$\frac{\{P \wedge b\}\ s\ \{I\},\ \{I \wedge b\}\ s\ \{I\},\ (I \wedge \sim b) \Rightarrow Q}{\{P\}\ \text{while } b \text{ do } s\ \{Q\}}\ ?$$

The rule is NOT valid. Proof:

```
{y≠17}              I: y=17
   while x>0 do
      y := 17
      x := x-1
   end_while
{y=17}
```

The three antecedents hold for the invariant y=17 but the consequent does not since the initial value of x may be ≤ 0 initially, in which case Q would not hold on termination.